UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Superresolución basada en deep-learning para la mejora de imágenes de territorio

Autor/es

CHRISTIAN STALIN BENALCÁZAR ADRIÁN

Director/es

JÓNATAN HERAS VICENTE

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Superresolución basada en deep-learning para la mejora de imágenes de territorio

Realizado por:

Christian Stalin Benalcázar Adrián

Tutelado por:

Jónathan Heras Vicente

Logroño, Junio de 2020

Resumen

La superresolución es un conjunto de técnicas y métodos que nos permiten mejorar la calidad de una imagen, además, de aumentar su resolución a partir de otra imagen de peor calidad.

Este trabajo de fin de grado consiste en el estudio de técnicas de superresolución, basadas en *deep learning*, con el fin de mejorar la resolución de imágenes de ortofotos aéreas del territorio riojano que surge desde IDERioja.

El foco principal del proyecto se centra en el estudio, entrenamiento y evaluación de diferentes modelos de redes neuronales, además, de la creación de un programa que nos permita la utilización de los distintos modelos generados durante este proyecto.

Abstract

Super-resolution is a class of techniques and methods which allows us to enhance the quality of an image, furthermore, to increasing its resolution from another image of poorer quality.

This final degree project consists of the study of super-resolution techniques, based on deep learning, to improve the resolution of aerial orthophoto images of La Rioja territory that arise from IDERioja.

The focus of the project lays on the study, training, and evaluation of different neural network models, in addition to creating a program which allow us to use the different models generated during this project.

Índice

1. Introducción 1
1.1. Antecedentes y justificación del proyecto 1
1.2. Superresolución 2
1.2.1. Resolución 3
1.2.2. Interpolación 4
1.2.3. Redes neuronales 4
2. Análisis 9
2.1. Alcance 9
2.2. Requisitos 9
2.3. Estudio de alternativas 10
2.3.1. Modelo generador 10
2.3.2. Funciones de pérdida 11
2.3.3. GAN 11
3. Planificación 13
3.1. Metodología 13
3.2. EDT 13
3.3. Cronograma e hitos 15
3.4. Plan de riesgos 16
4. Diseño 18
4.1. Arquitecturas 18
4.2. Creación de los modelos 19
4.2.1. Dataset de entrenamiento 19
4.2.2. Método de entrenamiento 21
4.2.3. Evaluación de los modelos 22
5. Entrenamiento 24
5.1. Estudio de plataformas y tecnologías 24
5.2. Desarrollo de los modelos 25
5.3. Programa 28
6. Evaluación 29
6.1. Resultados 29
6.2. Comparación visual 30
6.3. Experimentación 31
7. Seguimiento y control 33
7.1. Riesgos 33
7.2. Estimaciones de dedicación 33
7.3. Cronograma real 34
7.4. Entregables 35
8. Conclusiones 36
Bibliografía 37

1. Introducción

Dedicamos esta sección a explicar el contexto por el que surge este proyecto, además, de introducir conceptos que nos ayudarán a comprender mejor el problema que se nos plantea.

1.1. Antecedentes y justificación del proyecto

El **SIOSE** es el Sistema de Información sobre Ocupación del suelo de España cuyo objetivo es generar una base de datos de Ocupación del Suelo para toda España a escala de referencia 1:25000, integrando la información disponible de las Comunidades Autónomas y la Administración General del Estado [1].

Los principales objetivos de SIOSE son:

- Satisfacer las necesidades y los requerimientos de la Unión Europea, la Administración General del Estado y las comunidades autónomas en materia de ocupación del suelo.
- Integrar o recoger la información de las bases de datos de ocupación del suelo de la Administración General del Estado y de las comunidades autónomas.
- Hacer partícipes a las comunidades autónomas en el nivel de producción, control y gestión del SIOSE.
- Evitar las duplicidades y reducir costes en la generación de la información geográfica.

Asimismo, el **SIOSE de Alta Resolución** es un nuevo sistema de información construido por la integración de fuentes geoespaciales de alto detalle, y tiene como principal objetivo la integración, armonización y homogeneización de las mencionadas fuentes para seguir siendo un producto de referencia en la ocupación del suelo en España [2].

Por otra parte, desde el año 1990, el Gobierno de La Rioja viene utilizando la tecnología SIG (Sistema de Información Geográfica) para la gestión de su territorio, aplicando nuevas técnicas en la recolección de datos geográficos [3].

El Gobierno de La Rioja puso en marcha, en el año 2003/2004, el proyecto **IDERioja**, con la intención de poner la información geográfica al alcance de todos sus usuarios y de todos sus procedimientos de gestión [4].

IDERioja pone a nuestra disposición la siguiente información:

- Cartografía topográfica básica
- Ortofotos aéreas
- Cartografía temática
- Ficheros GPS para corrección diferencial

En este contexto, desde IDERioja surge la necesidad de aplicar nuevas técnicas de producción, ya que la resolución de las ortofotos aéreas que dispone IDERioja, en algunos casos, no es lo suficientemente buena.

En la imagen 1 podemos ver un ejemplo de una ortofoto aérea de La Rioja.

Imagen 1: Ortófono

Básicamente, cuando ampliamos ciertas zonas de la imagen, estas no se ven con la suficiente calidad, por lo que es necesario aplicar técnicas que mejoren su resolución.

Imagen 2: Comparación de una sección de una imagen con baja calidad y alta calidad

Mediante este trabajo pretendemos el estudio de técnicas de superresolución, con el fin de automatizar al máximo posible los proyectos de segmentación del territorio riojano, en concreto, con la utilización de redes neuronales y ortofotos aéreas obtenidas a través de IDERioja.

1.2. Superresolución

La superresolución es un conjunto de técnicas y algoritmos que permiten aumentar la resolución y mejorar el nivel de detalle de una imagen a partir de una imagen de más baja resolución [5].

A continuación, explicamos qué es la resolución de una imagen, además, de comparar técnicas más tradicionales, como la interpolación, con técnicas más modernas, como las redes neuronales.

1.2.1. Resolución

La **resolución** de una imagen es típicamente descrita en **PPI** (*Pixels Per Inch*, en inglés), la cual se refiere a cuantos píxeles se muestran en una pulgada de una imagen [6].

Una resolución más alta significa tener más píxeles por pulgada, lo que da como resultado más información de píxeles y una imagen más nítida y de mayor calidad.

Asimismo, imágenes con una resolución más baja significa tener menos píxeles por pulgada, y si esos pocos píxeles son muy grandes (normalmente cuando una imagen está estirada), pueden verse visibles como en la siguiente imagen.

Imagen 3: Ejemplo de imagen con baja resolución

Por ejemplo, una imagen que tiene una resolución de 600 ppi contendrá 600 píxeles dentro de cada pulgada de la imagen.

Por otra parte, la **dimensión** de una imagen digital se expresa en píxeles. Las dimensiones de una imagen se representan como *anchura x altura*. Por ejemplo, una imagen con 1000 píxeles de altura y 500 píxeles de anchura se representa como 500x1000.

La resolución va de la mano con la dimensión de una imagen, pero también va separada de ella. Lo que nos dice la resolución es cuantos píxeles se imprimirá en cada pulgada de papel. La resolución no significa nada hasta que la imagen está impresa [7].

Supongamos que tenemos una imagen con 3456 píxeles de anchura, 2304 píxeles de altura y 72 ppi de resolución, ¿cuánto de largo será la imagen si la imprimiésemos? Lo que tenemos que hacer es dividir las dimensiones de la imagen por la resolución. Después de calcularlo nos encontramos que nuestra imagen será 48 pulgadas (121,92 cm) de anchura y 32 pulgadas (81,28 cm) de altura. Si aumentamos la resolución, al imprimir, cada píxel de la imagen tiene que ser más pequeño.

Por ello, el único valor que importa en la pantalla son las dimensiones de la imagen [8]. Debido a este motivo, la resolución de una imagen digital se puede expresar mediante sus dimensiones.

Una vez explicado el concepto de resolución, vamos a presentar distintas formas en las que se puede aumentar la resolución de una imagen. Empezamos explicando la noción de interpolación.

1.2.2. Interpolación

La interpolación funciona usando información conocida para estimar valores en puntos desconocidos. La interpolación funciona en dos direcciones (para aumentar o reducir la resolución) y trata de conseguir la mejor aproximación de un píxel basado en los valores de los píxeles que lo rodean [9].

Algunos de los algoritmos de interpolación más conocidos son: el vecino más cercano, bilineal, bicúbica, etc. [10].

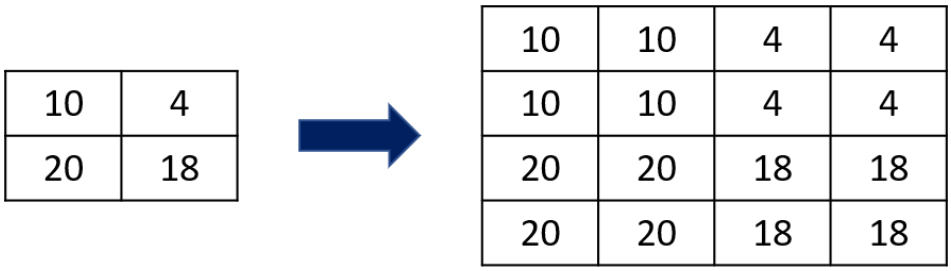

Imagen 4: Ejemplo de interpolación por vecino más cercano

A continuación, para ilustrar mejor el problema que nos encontramos con esta técnica, reducimos la resolución original de una imagen (3540x2430 píxeles) a su quinta parte, y luego comparamos la sección de esta imagen reducida (708x486 píxeles) a la que hemos aplicado interpolación bilineal para aumentar su resolución hasta su resolución original.

Imagen 5: imagen reducida (izquierda), aumentada (centro), original (derecha)

En la imagen 5 podemos observar cómo al aumentar la resolución de la imagen perdemos definición. Para resolver este problema, podemos utilizar técnicas más modernas como lo son las redes neuronales.

1.2.3. Redes neuronales

El **aprendizaje automático** (*Machine Learning*, en inglés) es una rama de la inteligencia artificial que proporciona a los ordenadores la capacidad de aprender, sin ser programadas

explícitamente. Los algoritmos de *machine learning* son unos maestros del reconocimiento de patrones, y son capaces de convertir una muestra de datos en un programa informático capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente [11].

El **aprendizaje profundo** (en inglés*, Deep Learning*) es una parte del *machine learning* que se basa en aprender varios niveles de representación, correspondiendo a una jerarquía de descriptores o factores o conceptos, donde los conceptos de nivel superior son definidos por conceptos de nivel inferior, y estos conceptos pueden ayudar a definir muchos conceptos de nivel superior.

El *deep learning* es parte de una familia más amplia de algoritmos de *machine learning* basados en aprender representaciones. Una observación (por ejemplo, una imagen) puede ser representada de muchas maneras (por ejemplo, un vector de píxeles), pero algunas representaciones hacen más sencillo aprender tareas de interés (por ejemplo, ¿es esta imagen una cara humana?) a partir de ejemplos, e investigaciones en esta área intentan definir mejoras en las representaciones y como aprenderlas [12].

Dentro del *deep learning*, las redes neuronales artificiales son **modelos** computacionales que intentan emular el comportamiento del cerebro humano.

Un **nodo (o neurona)** realiza una operación matemática sobre alguna entrada para calcular un resultado. La entrada de cualquier nodo es una suma ponderada de los resultados de la capa anterior [13]. Los pesos o coeficientes modifican el valor de las entradas.

Una **función de activación** determina como progresa el resultado anterior por la red neuronal.

A continuación, podemos observar mediante un gráfico el comportamiento de una neurona artificial.

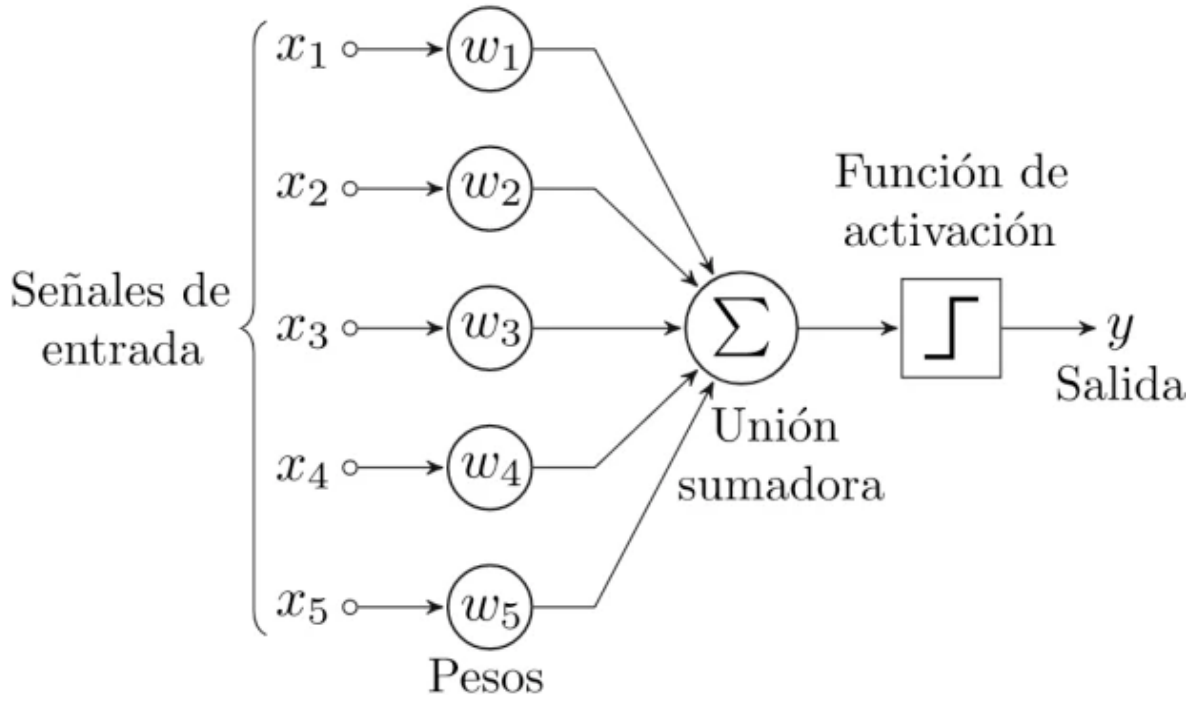

Imagen 6: Neurona o nodo [14]

La **función de pérdida** calcula el error entre el valor real y el predicho por la red neuronal. La red neuronal, mediante diferentes algoritmos, intenta minimizar el valor de esta función modificando los pesos de los nodos de la red neuronal.

Entrenar un modelo es un proceso que modifica sus pesos con el fin de que pueda, a partir de unos datos presentados en la entrada, generar una salida de manera que se reduzca la pérdida [15].

Denominamos **arquitectura** a la topología o estructura de una red neuronal artificial, es decir, a la organización y disposición de las neuronas en la red. Las neuronas se agrupan en unidades estructurales denominadas capas, y las capas se unen entre ellas formando redes neuronales.

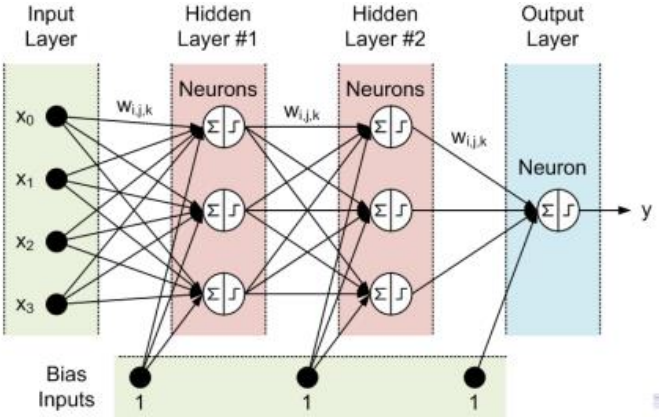

Imagen 7: Ejemplo de una arquitectura de red neuronal

Las **redes neuronales convolucionales** (CNN) son un tipo de red neuronal artificial que permiten *ver* al ordenador. En estas redes se trabaja con volúmenes. Un volumen puede ser, por ejemplo, una imagen.

Un filtro es una pequeña matriz que se coloca y se desplaza por una imagen. Una convolución es una operación matemática que se aplica a una capa de la red neuronal.

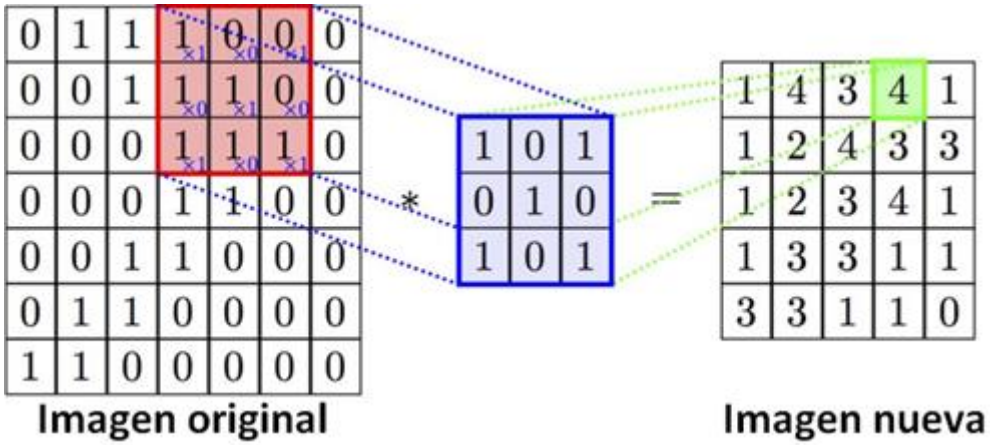

Imagen 8: Ejemplo de una convolución

El resultado es una capa convolucional, que es el componente principal de una CNN. El objetivo de una CNN es aprender los filtros.

Una CNN está compuesta por varias capas convolucionales y otros tipos de capas como: capas de activación, capas de pooling, capas completamente conectadas, etc.

Aplicado a nuestro proyecto, existen varias arquitecturas de redes neuronales aplicadas a superresolución, pero que pueden ser atribuidas a los siguientes marcos de trabajo.

Estos marcos de trabajo están basados en las operaciones de interpolación utilizadas y su localización en el modelo [16].

La imagen 9 muestra el primer marco de trabajo, *Pre-Upsampling*, donde primero aumentamos una imagen para luego realizar convoluciones.

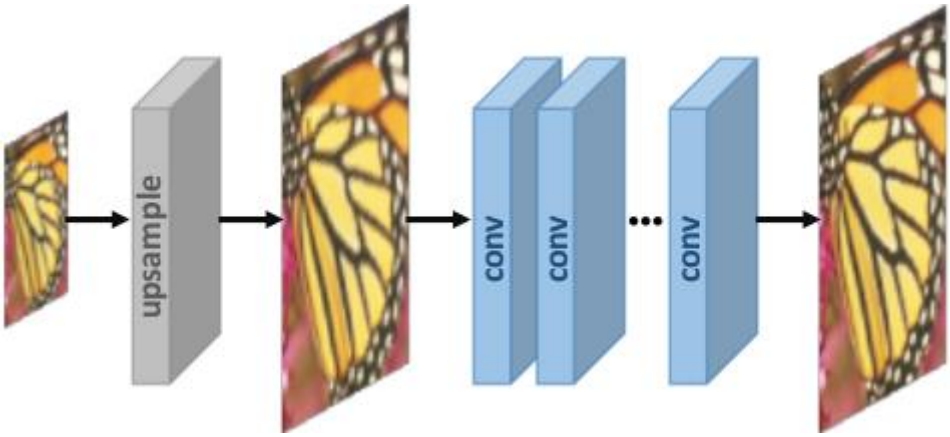

Imagen 9: Una típica red *Pre-Upsampling*

La imagen 10 muestra el segundo marco de trabajo, *Post-Upsampling*, donde aumentamos la imagen en la última capa del modelo.

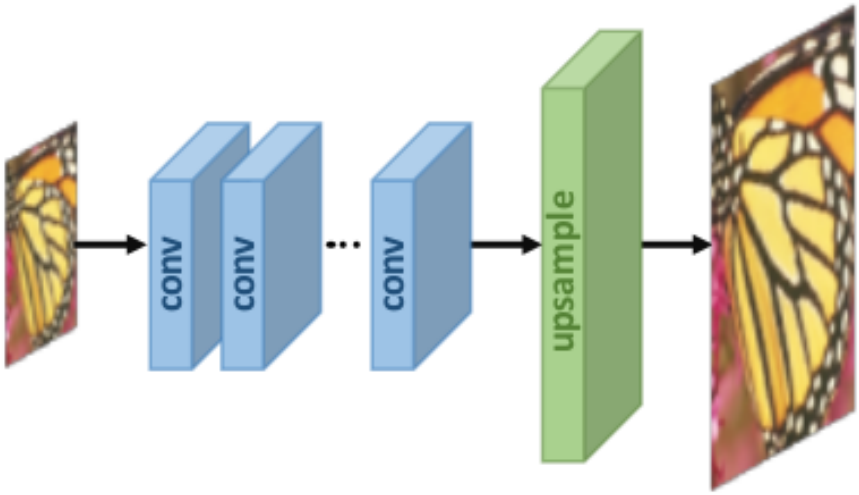

Imagen 10: Una típica red *Post-Upsampling*

La imagen 11 muestra el tercer marco de trabajo, *Progressive Upsampling*, donde usamos una cascada de CNNs para reconstruir progresivamente las imágenes de alta resolución en factores pequeños en cada paso.

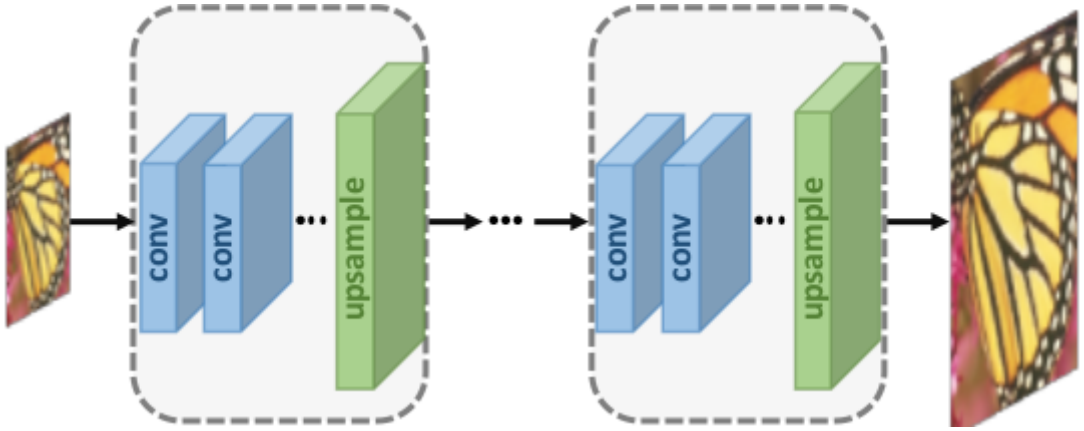

Imagen 11: Una típica red *Progressive Upsampling*

La imagen 12 muestra el cuarto y último marco de trabajo, *Iterative Up-and-Down Sampling*, donde alternamos entre proceso que aumentan y reducen la imagen.

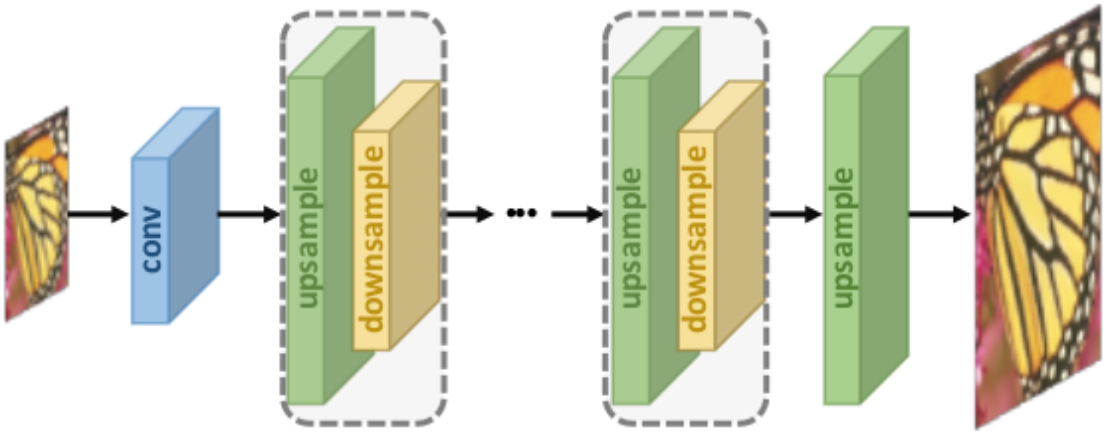

Imagen 12: Una típica red *Iterative Up-and-Down Sampling*

Los modelos bajo este último marco pueden explotar mejoras las relaciones entre los pares de imágenes de baja y alta resolución y, por tanto, pueden proporcionar resultados de reconstrucción de mayor calidad.

Las superresolución basada en redes neuronales convolucionales es útil principalmente porque trata problemas encontrados en las técnicas clásicas de superresolución. Estas técnicas clásicas no son capaces de generar detalles finos en la imagen y, al mismo tiempo, no pueden tratar con artefactos de compresión u otros defectos de imagen. Los modelos entrenados para tareas de superresolución también se pueden usar para reparar defectos de la imagen, como eliminar compresiones y píxeles corruptos [17].

En este contexto, en este proyecto utilizaremos modelos de redes neuronales basados en este cuarto marco de trabajo.

2. Análisis

Dedicamos esta sección a presentar el alcance del proyecto y sus requisitos, además, de realizar un estudio de alternativas de técnicas actuales de superresolución con redes neuronales.

Introducimos esta sección antes que la planificación, ya que el análisis nos ayudará a comprender mejor la planificación del proyecto.

2.1. Alcance

El alcance de este proyecto consiste en aplicar técnicas de superresolución, basadas en *deep learning,* a ortofotos aéreas de La Rioja obtenidas desde IDERioja con el fin de aumentar su resolución. Para ello estudiaremos distintos modelos de superresolución y compararemos su rendimiento. La imagen 13 muestra un esquema del alcance del proyecto.

Imagen 13: Alance del proyecto

2.2. Requisitos

A continuación, presentamos los requisitos que presenta este proyecto.

Requisitos funcionales:

- Estudio de las distintas alternativas de técnicas de superresolución basadas en *deep learning*.
- Construir diferentes modelos de superresolución. Los modelos podrán aumentar en cinco veces la resolución de una ortofoto.
- Evaluar los modelos construidos y seleccionar aquel que produce mejores resultados, atendiendo a diversas métricas o el tiempo de inferencia.
- Desarrollo de una aplicación por línea de comandos para utilizar el modelo.

Requisitos no funcionales:

- Se utilizará Python como lenguaje de programación.
- Se trabajará con imágenes JPG.
- Todos los modelos desarrollados deben estar en un repositorio de GitHub.

2.3. Estudio de alternativas

En la actualidad, las técnicas de superresolución basadas en aprendizaje profundo han demostrado tener una mayor superioridad comparadas a otras técnicas [18]. Por este motivo, resulta más adecuado para este proyecto la utilización de este tipo de técnicas.

2.3.1. Modelo generador

Denominamos **generador** al modelo que aumenta la resolución de una imagen. Dentro de los posibles modelos generadores, la opción más interesante es una **U-Net**, debido a los buenos resultados obtenidos en otros problemas de visión por ordenador.

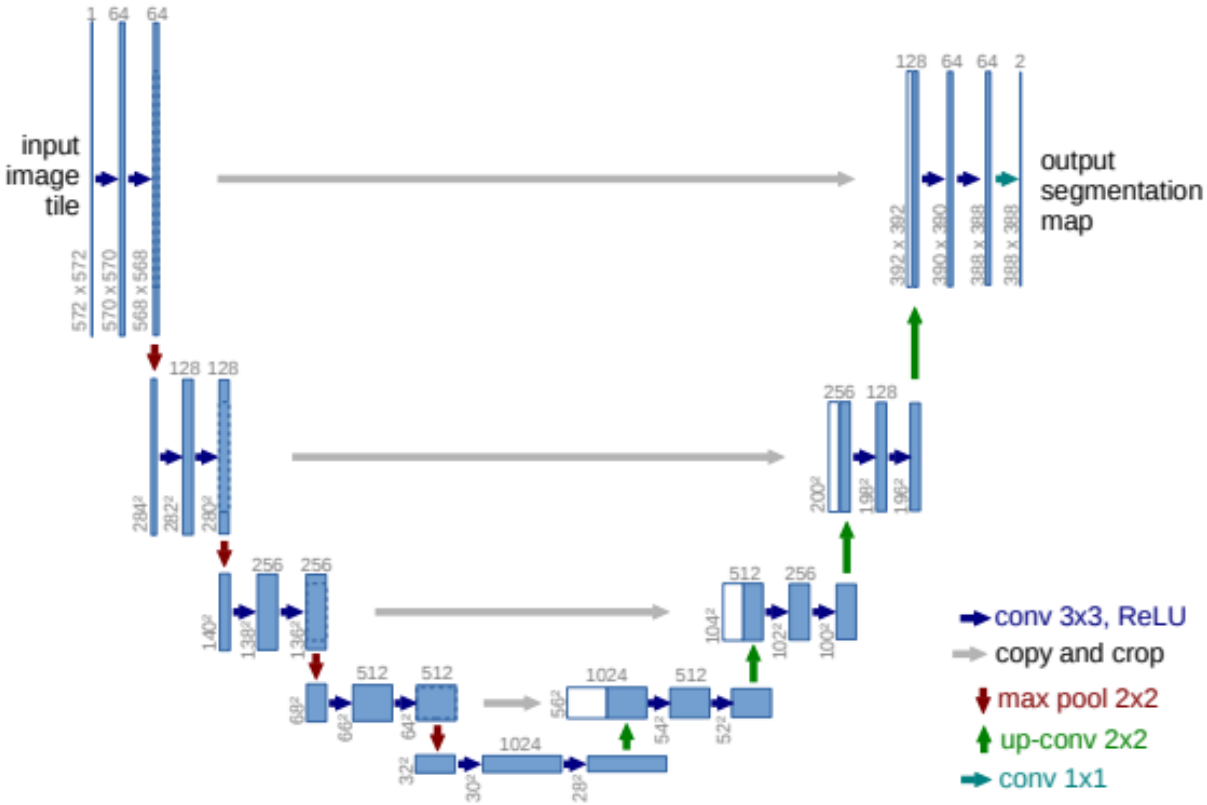

Imagen 14: U-Net utilizada para la segmentación de imágenes biomédicas [19]

En este modelo generador, que tiene un camino de reducción (*downsampling*) seguido de otra de aumentación (*upsampling*), denominamos **codificador** (*encoder*) al primer camino y **decodificador** (*decoder*) al segundo camino respectivamente.

Antes de U-Net, el decodificador no producía muy buenos resultados, porque resultaba difícil de reconstruir un mapa de descriptores de, por ejemplo, 572x572 (la entrada y salida del modelo) a partir de un mapa de descriptores de, por ejemplo, 28x28 (la capa inferior del codificador).

U-Net concatena un mapa de descriptores del codificador a otro mapa de descriptores del decodificador situado en el mismo nivel. Por ejemplo, un mapa de descriptores de las primeras capas se concatena a las últimas capas del modelo. De este modo, estas conexiones ayudan al decodificar a mejorar los detalles en la tarea de reconstrucción.

2.3.2. Funciones de pérdida

Dentro de las técnicas de superresolución con redes neuronales, podemos encontrarnos diferentes modelos según la función de pérdida que utilicen.

En primer lugar, nos encontramos con **Píxel *MSE*** (*Mean Squared Error*, en inglés), la cual es una función de pérdida sencilla que calcula el error medio cuadrático entre cada píxel de la imagen real y la predicha por el modelo [20].

El inconveniente que presenta está función de pérdida es que la diferencia entre los píxeles es muy baja, porque la mayoría de los píxeles que predice el modelo están muy cerca de su valor real.

En segundo lugar, nos encontramos con la función ***perceptual loss*** [21]**,** la cual calcula el error comparando dos imágenes basadas en representaciones de alto nivel de modelos CNN pre-entrenados. La función es usada para comparar diferencias de alto nivel, como discrepancias de estilo, entre imágenes.

En otras palabras, tanto la imagen real como la predicha se pasan a través de un modelo pre-entrenado y se calcula la distancia euclídea entre los dos mapas de descriptores resultantes (al mismo nivel). La función *perceptual loss* funciona sumando todos los errores cuadrados entre todos los píxeles y tomando su media, en contraste con funciones de pérdida por píxeles, que suman todos los errores absolutos entre píxeles [9].

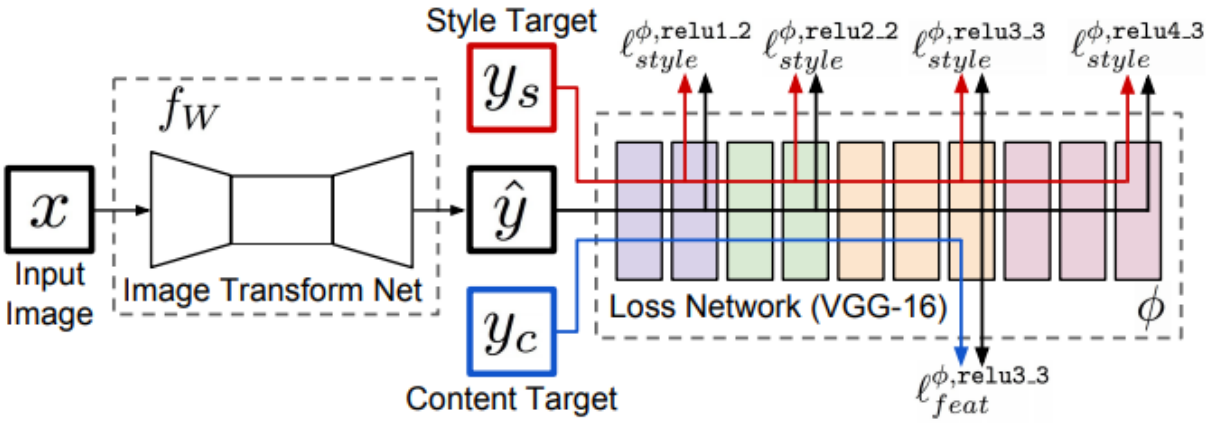

Imagen 15: Ejemplo de perceptual loss (Loss Network)

En cuanto al modelo que se utiliza para calcular *perceptual loss*, se suele utilizar VGG-16 [22] el cual es un modelo que alcanzó una precisión del 92,5% de testeo en *ImageNet* [23] (un banco de imágenes con más de 14 millones de ejemplos pertenecientes a más de 1000 clases).

2.3.3. GAN

Por otra parte, como alternativa, nos encontramos con una **GAN** *(Generative Adversarial Networks*), la cual es un tipo de red neuronal que utiliza como función de pérdida una llamada a otro modelo [24].

Las GANs se aplican en nuestro caso del siguiente modo: tenemos una imagen de baja calidad y un modelo generador. La predicción del modelo no es mala, aunque tampoco es buena.

Además, tenemos la imagen de alta resolución original y podemos compararlas entre sí con Píxel MSE.

Podemos mejorar al generador entrenando otro modelo, al que llamaremos discriminador o crítico. Este modelo toma la imagen predicha y la original, y trata de distinguir cuál es cuál.

Si la función de pérdida es "¿estoy engañando al crítico?", el generador será capaz de predecir imágenes que el crítico no pueda distinguir entre la imagen real o la predicha.

Al principio, el crítico es un buen modelo, porque las imágenes son muy fáciles de distinguir. Así que entrenamos más al generador. En cambio, el crítico empeora ya que le resulta más difícil distinguirlas. Así que entrenamos nuevamente al crítico. De este modo, estaremos jugando al ping-pong con estos modelos hasta que el generador sea lo suficientemente bueno. Esto es un ejemplo de GAN.

Sin embargo, las GANs presentan un problema, son muy difíciles de entrenar. Además, los errores que devuelven los modelos carecen de significado. No podemos esperar que los errores del generador y crítico disminuyan, ya que a medida que el generador mejora, se vuelve más difícil para el crítico, y luego, a medida que el crítico mejora, se vuelve más difícil para el generador. Así que los errores permanecen casi iguales. En la imagen 16 podemos ver cómo evoluciona los valores de pérdida durante el entrenamiento.

epoch	train_loss	gen_loss
1	2.071352	2.025429
2	1.996251	1.850199
3	2.001999	2.035176
4	1.921844	1.931835
5	1.987216	1.961323
6	2.022372	2.102732

Imagen 16: Errores de un modelo GAN

De esta forma, en este proyecto, utilizaremos U-Net como modelo generador y construiremos modelos diferentes con las dos funciones de pérdida anteriores y con una arquitectura GAN.

3. Planificación

En esta sección dedicamos a exponer la metodología elegida para este proyecto, la estructura de descomposición de trabajo, cronograma e hitos y, por último, el plan de riesgos.

3.1. Metodología

Tras un breve análisis de las distintas metodologías existentes, hemos decidido utilizar la metodología en cascada debido a que se trata de un proyecto bien definido. Sin embargo, nos diferenciamos de la metodología original porque en este proyecto no existe implementación, no implementamos modelos, sino que entrenaremos modelos ya existentes. Además, existe iteración en las fases de entrenamiento y evaluación de la metodología, ya que, para encontrar el mejor modelo tenemos que evaluarlo y estudiar los diferentes modelos hasta que encontremos un resultado satisfactorio.

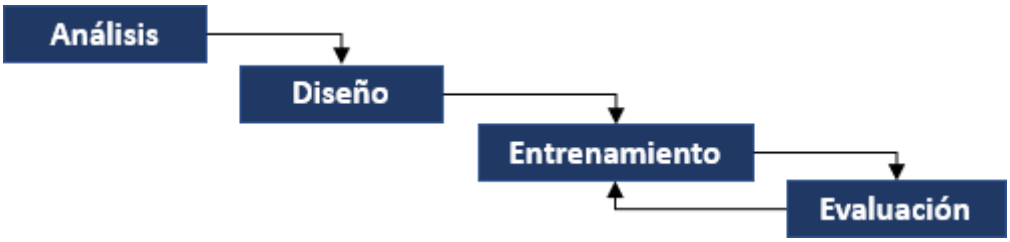

Imagen 17: Metodología del proyecto

3.2. EDT

La imagen 18 muestra la EDT (estructura de descomposición de trabajo) del proyecto.

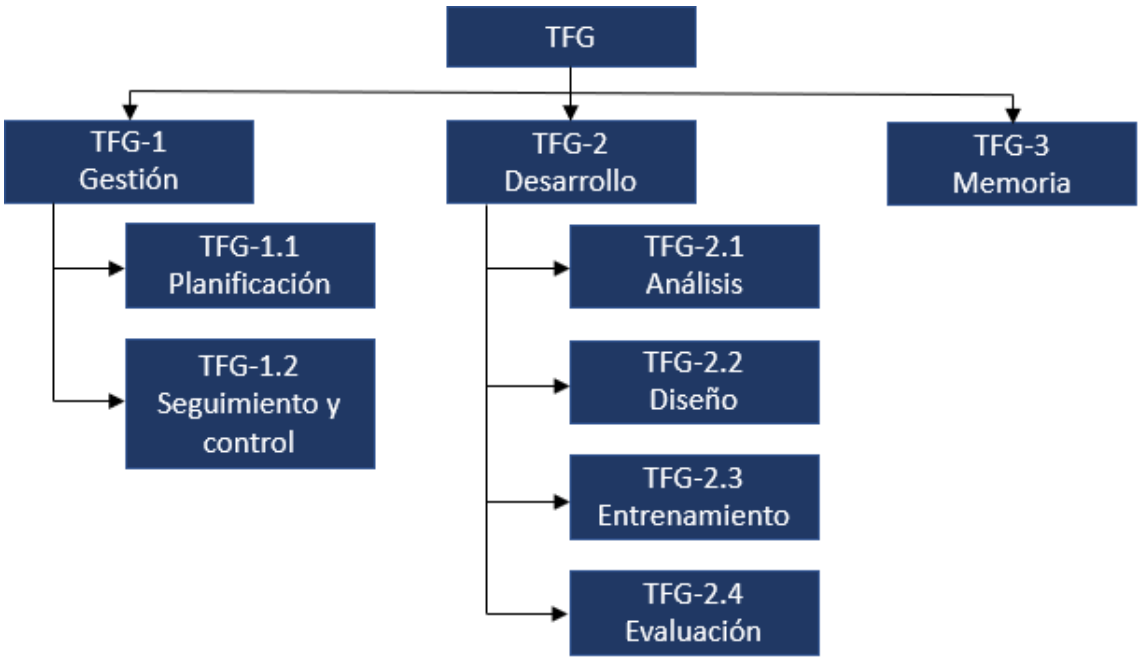

Imagen 18: EDT del proyecto

La tabla 1 muestra el diccionario de la EDT. Además de un código y un título, se describe su contenido.

ID	Título	Descripción
TFG-1	Gestión del proyecto	
TFG-1.1	Planificación	Contiene la metodología, EDT, cronología y el plan de dedicaciones y riesgos del proyecto.
TFG-1.2	Seguimiento y control	Contiene el seguimiento del proyecto.
TFG-2	Desarrollo	
TFG-2.1	Análisis	Contiene el alcance, los requisitos y un análisis de alternativas de métodos existentes.
TFG-2.2	Diseño	Contiene las arquitecturas de los modelos y cómo se crearán.
TFG-2.3	Entrenamiento	Contiene un estudio de las diferentes plataformas existentes, las tecnologías que se usarán, el desarrollo de la construcción del dataset, los modelos y el programa.
TFG-2.4	Evaluación	Se comparan los resultados obtenidos.
TFG-3	Memoria	Es el presente documento. Contiene las diferentes etapas que forman el proyecto.

Tabla 1: Diccionario de la EDT

La tabla 2 muestra la estimación en horas para cada paquete de trabajo de la EDT .

Paquete de trabajo		Horas
TFG-1.1	Planificación	30
TFG-1.2	Seguimiento y control	20
TGF-2.1	Análisis	30
TFG-2.2	Diseño	30
TFG-2.3	Entrenamiento	120
TFG-2.4	Evaluación	10
TFG-3	Memoria	60
TFG		300

Tabla 2: Plan de dedicaciones del proyecto

3.3. Cronograma e hitos

La tabla 3 muestra el diagrama de Gantt del TFG. En él se distribuye los paquetes de trabajo a lo largo de las 16 semanas de duración.

Paquete de trabajo		S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16
TFG-1.1	Planificación																
TFG-1.2	Seguimiento y control																
TFG-2.1	Análisis																
TFG-2.2	Diseño																
TFG-2.3	Implementación																
TFG-2.4	Evaluación																
TFG-3	Memoria																
TFG																	

Tabla 3: Diagrama de Gantt del proyecto

La tabla 4 muestra los hitos a lo largo del proyecto. Cada paquete de trabajo tiene un hito que corresponde con el cierre de esa fase.

Hito	Fecha
Inicio TFG	10/02/2020
Cierre planificación	21/02/2020
Cierre análisis	06/03/2020
Cierre diseño	20/03/2020
Cierre implementación	08/05/2020
Cierre evaluación	15/05/2020
Cierre seguimiento y control	22/05/2020
Cierre memoria	22/05/2020
Fin TFG	28/05/2020
Entrega	22/06/2020
Defensa	13/07/2020

Tabla 4: Hitos del proyecto

3.4. Plan de riesgos

La tabla 5 muestra el plan de comunicaciones con el tutor que ejercerá como cliente.

Tipo	Informar	Pedir información	Llegar a acuerdos
Síncrona	Reunión cada dos semanas, de lunes a jueves de 09:00 a 20:00	Reunión, de lunes a jueves de 09:00 a 20:00	Reunión, de lunes a jueves de 09:00 a 20:00
Asíncrona	Correo electrónico, de 09:00-21:00	Correo electrónico, de 09:00-21:00	Correo electrónico, de 09:00-21:00

Tabla 5: Plan de comunicación con el tutor

Las tablas 6 a 9 muestran los riesgos que pueden aparecer a lo largo del proyecto.

Riesgos con el tutor			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Ausencia del tutor.	No se puede prevenir.	Mantener una comunicación constante.	Si es necesario, apoyarse en otros tutores.

Tabla 6: Riesgos con el tutor

Riesgos con el tamaño del producto			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Retrasos.	Realizar una buena estimación en horas del proyecto.	Cumplir con los hitos del proyecto.	Modificar el alcance u otro punto del proyecto con el permiso del tutor.

Tabla 7: Riesgos con el tamaño del producto

Riesgos con la tecnología			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Problemas con la plataforma.	Realizar un estudio de las diferentes plataformas.	No se puede minimizar el riesgo.	Cambiar de plataformas si es posible.

Tabla 8: Riesgos con la tecnología

Riesgos con el desarrollo del producto			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Pérdida del código y memoria.	Realizar varias copias tanto en local como en la nube.	No se puede minimizar el riesgo.	Recuperación de las copias de seguridad.

Tabla 9: Riesgos con el desarrollo del producto

4. Diseño

Dedicamos esta sección a explicar las arquitecturas de las redes neuronales que utilizaremos en este proyecto y cómo realizaremos la creación de los modelos.

4.1. Arquitecturas

La imagen 19 muestra una red neuronal que utilizará como generador U-Net y Píxel MSE como función de pérdida.

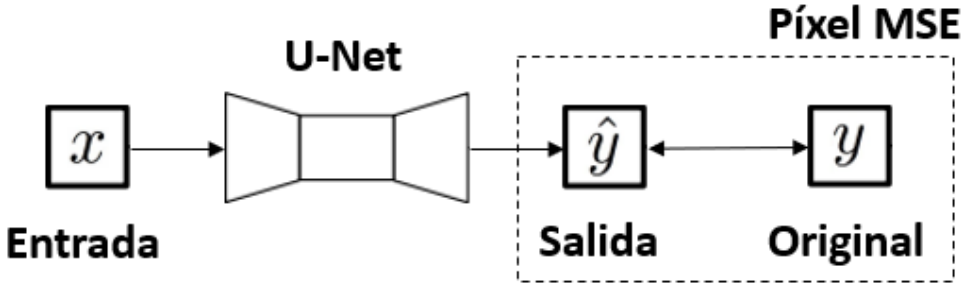

Imagen 19: Red neuronal con Píxel MSE

La imagen 20 muestra una red neuronal que utilizará como generador U-Net y como función de pérdida *perceptual loss* con VGG-16.

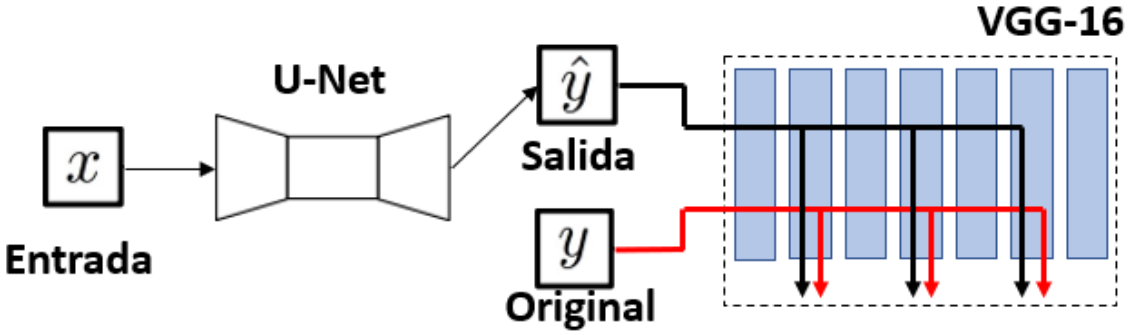

Imagen 20: Red neuronal con Perceptual Loss

La imagen 21 muestra una red neuronal que tendrá una arquitectura GAN.

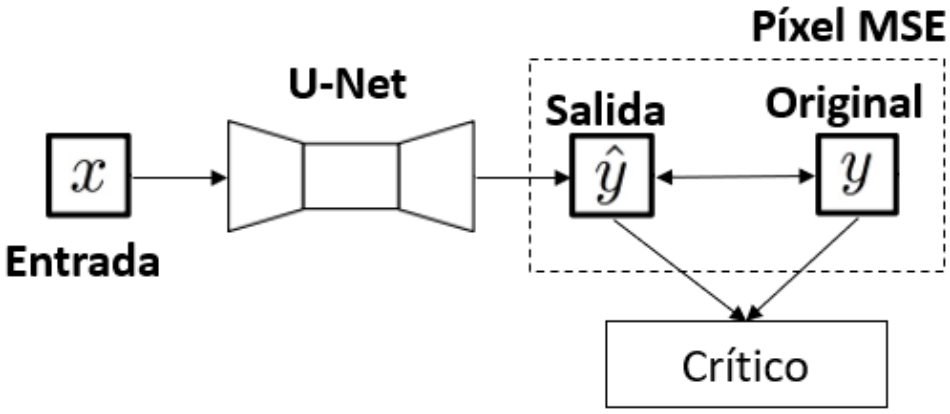

Imagen 21: Red neuronal GAN

4.2. Creación de los modelos

A continuación, explicaremos cómo crearemos el dataset de entrenamiento, el método que emplearemos para el entrenamiento de los modelos y cómo evaluaremos los modelos.

4.2.1. Dataset de entrenamiento

IDERioja pone a nuestra disposición ortofotos en color de los años 2000, 2004, 2006, 2009, 2012, 2014 y 2017. Cada año tiene un conjunto de 741 imágenes, haciendo un total de 5187 imágenes disponibles para la creación del dataset de entrenamiento.

Sin embargo, algunas de esas imágenes no están completas en su totalidad, por lo que se ha decidido eliminarlas del dataset. Esta decisión nos proporcionará ciertas ventajas como veremos más adelante.

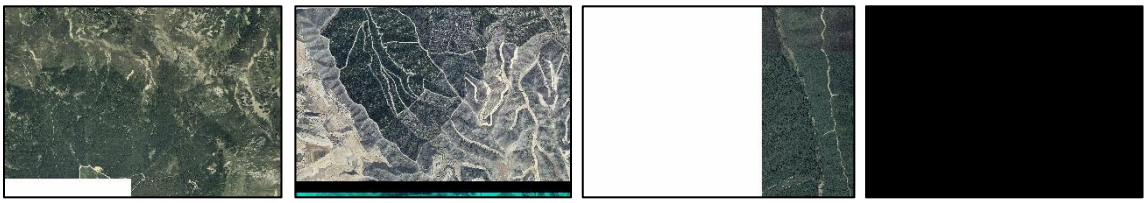

Imagen 22: Algunas de las imágenes eliminadas del dataset

En total disponemos de 5067 imágenes. Cada imagen tiene una resolución diferente, aunque estas están alrededor de 3560x2450 píxeles.

Es importante destacar que en la creación del dataset, es necesario que este tenga los detalles que se desean mejorar con el modelo, ya que el modelo no será capaz de aprender a mejorar los aspectos que no están presentes en el dataset.

Siendo así, en nuestro proyecto, crearemos dos datasets. En ambos casos, reduciremos la resolución de la imagen original, con interpolación bilineal, a una que tenga una altura de 256 píxeles, pero con la misma relación de aspecto. Es decir, si la imagen original tiene una resolución de 3500x2400, la nueva imagen tendrá una resolución de 373x256 píxeles, manteniendo así la relación de aspecto 1:1.457 en la nueva imagen, en la medida de lo posible.

Al trabajar con imágenes JPG debemos tener en cuenta lo siguiente: JPG es un algoritmo de compresión por perdida. En este tipo de compresión, algunos detalles se pierden. No perdemos píxeles sino detalles como el color [25].

La calidad JPG nos permite seleccionar el nivel de compresión de una imagen. Una calidad JPG 10 es de muy baja calidad, mientras que una imagen con calidad JPG 80 es de alta calidad.

De este modo, en el primer dataset, la calidad JPG de cada imagen será de 75. En cuanto al segundo dataset, guardaremos cada imagen nueva con una calidad JPG aleatoria entre 10 y 80, así el modelo creado será más robusto ante imágenes que presenten distinto nivel de compresión.

Imagen 23: Ortofoto con calidad JPG 10

Imagen 24: Ortofoto con calidad JPG 75

Imagen 25: Ortofoto con calidad JPG 80

Este segundo dataset, lo utilizaremos para producir un modelo más robusto ante imágenes con muy baja calidad, una vez hayamos seleccionado el modelo que produce los mejores resultados con el primer dataset.

Por otra parte, también realizaremos *data augmentation* [26]. Esta técnica nos permite entrenar al modelo con transformaciones de las imágenes (aumentando así el tamaño del dataset), realizando pequeñas modificaciones sobre las imágenes originales, como rotarlas, voltearlas, escalarlas, cambiar la iluminación, etc.

Durante el entrenamiento, se suele dividir el dataset de entrenamiento en dos conjuntos: uno para el entrenamiento y otro para test. Esta división se realiza para comprobar que nuestro modelo no se sobreajuste. Es decir, queremos evitar que, en lugar de generalizar el problema, el modelo aprenda a memorizar los datos de entrada.

Este sobreajuste lo podemos empezar a ver cuándo evaluamos el modelo con el propio conjunto de entrenamiento, ya que cuando comparemos los valores de pérdida de ambos conjuntos, si el valor de pérdida del conjunto de entrenamiento empieza a disminuir significativamente frente al valor de pérdida del conjunto de test, esto es un indicio de que el modelo está empezando a sobreajustarse.

Lo que buscamos es que el tanto el error de test como el de entrenamiento disminuyan, además, de que el error de test sea ligeramente mayor que el error de entrenamiento.

En cuanto a la división del dataset de entrenamiento, esta suele ser del 80% y 20% del dataset original para el conjunto de entrenamiento y test respectivamente.

Por último, en cuanto al dataset de test, lo crearemos reduciendo la resolución original de la imagen a su quinta parte y con una calidad JPG 75. En total utilizaremos 500 imágenes.

4.2.2. Método de entrenamiento

Entrenaremos los algoritmos de forma progresiva en factores de 2, es decir, primero entrenaremos el modelo para que aumente la resolución de 256x256 píxeles a 512x512, y luego de 512x512 a 1024x1024, y así hasta 2048x2048.

El porqué de esta decisión es muy sencillo, es más rápido entrenar con imágenes pequeñas al principio y después aumentar la resolución progresivamente. Aumentar la resolución de una imagen desde 256x256 píxeles hasta 512x512, es una tarea más fácil que realizarla con una resolución más grande. Esto se llama redimensionamiento progresivo (*progressive resizing*, en inglés). Además, esto ayuda al modelo a generalizar mejor porque ve más imágenes diferentes y hace que sea menos probable que el modelo se sobreajuste [27].

Asimismo, la razón por la que entrenaremos al modelo con todas las imágenes cuadradas y con la misma resolución, es por eficiencia. La GPU realizará los cálculos más eficientemente si todas las imágenes tienen el mismo tamaño.

Por otra parte, un aspecto importante del modelo es la elección de un buen *learning rate*. Este parámetro de ajuste del modelo controla cuanto estamos ajustando los pesos de nuestra red respecto a la función de pérdida [28]. Si este parámetro es bajo, nuestra red puede tardar

mucho en aprender, en cambio, si este es alto, cada paso puede sobrepasar el mínimo de la función y nunca obtendremos un valor de pérdida aceptable. Peor aún, un *learning rate* alto puede llevarnos a que el valor de pérdida aumente progresivamente [29].

Por último, utilizaremos *transfer learning* [30]**.** Esta es una técnica donde usamos un modelo entrenado en un dataset grande (generalmente *ImageNet* en visión por computador) y luego lo adaptamos a nuestro propio dataset. La idea es que el modelo ya ha aprendido a reconocer patrones en el dataset, entonces nos podemos beneficiar de este conocimiento, especialmente si nuestro dataset es pequeño.

Esta técnica ha demostrado en una amplia gama de tareas que *transfer learning* casi siempre produce buenos resultados [31].

En la práctica, debemos cambiar la última parte del modelo para adaptarlo a nuestro problema. Cuando entrenemos nuestro modelo, primero congelaremos los pesos del modelo y entrenaremos solo la cabeza (el decodificador en el caso de U-Net) para adaptarlo a nuestro problema, para luego descongelar los pesos y realizar *fine-tuning* con todo el modelo.

Fine-tuning es un proceso donde los pesos de un modelo son ajustados para que encajen mejor con nuestra tarea. Esta técnica nos ayuda a entrenar más fácilmente el modelo.

En nuestro proyecto, utilizaremos el modelo ResNet50 [32] para *transfer learning*. Este modelo es usado como el codificador en U-Net [27].

4.2.3. Evaluación de los modelos

A priori no podemos comparar los modelos por su error de validación, ya que cada modelo utiliza una función de pérdida diferente, así que vamos a evaluarlos con las siguientes métricas: PSNR (*Peak Signal-to-Noise Ratio*) y SSIM (*Structural Similarity*) [33].

PSNR mide la calidad de reconstrucción de una transformación con pérdida. Para superresolución, PSNR es definido mediante el valor máximo del píxel (denotado como *L*) y el error medio cuadrático (MSE) entre las imágenes. Dada la imagen original *I* con *N* píxeles y la predicha $\hat{I}$, PSNR entre *I* y $\hat{I}$ se define como:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(I(i) - \hat{I}(i))^2,$$

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{L^2}{\mathrm{MSE}}\right).$$

Imagen 26: PSNR

Para imágenes RGB de 8 bits, *L* es igual a 255. Dado que PSNR solo se preocupa de las diferencias entre píxeles, no representa de manera adecuada la calidad perceptiva del sistema visual humano (HSV).

Asimismo, dado que MSE es el denominador, si el error es alto, el valor de PSNR será bajo. Un valor alto de PSNR indica una imagen con mejor calidad.

Por otro lado, **SSIM** mide la similitud estructural entre imágenes, basadas en tres comparaciones relativamente independientes: la luminancia, el contraste y la estructura. En resumen, la fórmula SSIM puede mostrarse como un producto ponderado de la comparación de luminancia, contraste y estructura.

$$\mathrm{SSIM}(I, \hat{I}) = [\mathcal{C}_l(I, \hat{I})]^{\alpha}[\mathcal{C}_c(I, \hat{I})]^{\beta}[\mathcal{C}_s(I, \hat{I})]^{\gamma},$$

Imagen 27: SSIM

En la formula anterior, las comparaciones de luminancia, contraste y estructura son denotados como $C_l(I, \hat{I})$, $C_c(I, \hat{I})$, $Cs(I, \hat{I})$ respectivamente. Alpha, beta y gamma son parámetros de control para ajustar la importancia relativa de cada producto.

En cualquier caso, dado que SSIM evalúa la calidad de reconstrucción desde la perspectiva del sistema visual humano, cumple mejor los requisitos de calidad perceptiva.

El índice SSIM resultante es un valor decimal entre -1 y 1, y el valor 1 solo es accesible en el caso de que los dos conjuntos de datos sean idénticos y, por lo tanto, indica una similitud estructural perfecta. Un valor de 0 indica que no hay similitud estructural [34].

De esta forma, utilizaremos tanto PSNR y SSIM para evaluar los modelos.

Asimismo, evaluaremos a los modelos según el tiempo de inferencia que tarda cada uno. De esta forma mediremos la eficiencia de cada modelo.

5. Entrenamiento

Esta sección la dedicamos para presentar un estudio de las diferentes plataformas que podemos usar durante esta etapa del proyecto, las tecnologías, el desarrollo de la construcción del dataset, los diferentes modelos y el programa por línea de comandos.

5.1. Estudio de plataformas y tecnologías

En este proyecto vamos a utilizar como lenguaje de programación a Python, debido a que la mayor parte de algoritmos de superresolución están en este lenguaje. Asimismo, utilizaremos como librería a fastai [35], ya que nos proporciona la implementación de U-Net (otras librerías famosas como Keras o TensorFlow no lo tienen implementado directamente) y otras funciones que facilitarán el entrenamiento de los modelos.

En cuanto al entorno, vamos a utilizar Jupyter Notebook o su generación más reciente JupyterLab [36]. Ambos entornos informáticos interactivos basados en la web nos permiten crear documentos donde podemos escribir y ejecutar código Python.

Debido a que se trata de un proyecto que requiere de una gran potencia de procesamiento, las opciones más adecuadas se encuentran en la nube. A continuación, listamos algunas de las plataformas que podemos utilizar durante el proyecto, junto con sus ventajas e inconvenientes.

- **Google Colab:** Esta plataforma, de uso gratuito, nos proporciona recursos como 1 Core de CPU, alrededor de 16 GB de RAM y tarjetas gráficas como Nvidia T4 o Nvidia P100 de 16 GB. El principal inconveniente que presenta esta plataforma es que la disponibilidad de recursos no está garantizada, además, solo tiene una duración de hasta 12 horas, tras la cual debemos reiniciar el entorno [37].
- **Google Cloud**: Esta plataforma nos permite crear instancias de máquinas virtuales con JupyterLab, en la cual podemos adaptar el hardware si nuestras necesidades lo exigen. De igual manera, la disponibilidad de ciertos recursos no está garantizada en todas las regiones. Esta plataforma nos proporciona una prueba gratuita de 12 meses con 300$ de créditos para usar los servicios de esta plataforma [38].
- **AWS:** Esta plataforma nos permite ejecutar notebooks de forma gratuita mediante los servicios Amazon EC2 o Amazon SageMaker. Amazon nos proporciona solo algunas instancias gratuitamente de todas las disponibles durante un periodo de tiempo [39]. Por otra parte, AWS Educate nos permite ampliar la gama de posibilidades gracias a su programa para estudiantes [40].

De este modo, en este proyecto, empezaremos utilizando Google Colab junto con Google Drive debido al conocimiento de esta plataforma y herramienta respectivamente. Si fuese necesario, la siguiente plataforma a utilizar sería Google Cloud, de igual manera, debido al conocimiento de esta.

5.2. Desarrollo de los modelos

El entrenamiento empezó con Google Colab. De manera inmediata, era evidente la necesidad de ampliar la capacidad de almacenamiento de Google Drive, porque subir todas las imágenes nos consumía más de los 15 GB gratis disponibles. Por ello, decidimos pagar 2 euros al mes para ampliar la capacidad hasta 100 GB de almacenamiento, ya que el coste era muy bajo.

El primer modelo creado fue con *perceptual loss*. En un primer intento, realizamos el entrenamiento con todas las imágenes sin haber eliminado las imágenes incompletas del dataset para comprobar si afectaba o no al entrenamiento del modelo. Enseguida surgió la necesidad de disminuir el número de las imágenes del dataset, ya que se exigía más memoria tanto en la tarjeta gráfica como en la RAM de la que disponíamos.

La disminución del dataset, además de resolver el problema anterior, producía una mejora en la rapidez del entrenamiento.

Asimismo, el entrenamiento en las etapas posteriores del modelo tardaba más de las 12 horas disponibles de ejecución con Google Colab. Además, en un par de ocasiones el entrenamiento se paraba paró porque Google no podía seguir proporcionándonos una tarjeta gráfica. Por este motivo, realizamos un cambio de plataforma a Google Cloud.

En Google Cloud, empleamos una máquina virtual con 4 vCPUs, 15 GB de memoria RAM y una tarjeta gráfica Nvidia T4 o Nvidia P100 con 16 GB de memoria (siendo esta última mucho más rápida). La diferencia de rendimiento era muy superior, tanto en las fases iniciales del entrenamiento como en las fases posteriores.

De igual manera, la utilización de ResNet50 nos llevaba a un consumo muy por encima de la memoria de la tarjeta gráfica. Debido a este motivo, decidimos cambiar a ResNet34, el cual es un modelo más sencillo y, por tanto, consume menos recursos.

Por otra parte, en todos los modelos no pudimos realizar el último paso de aumentar la resolución de 1024x1024 píxeles hasta 2048x2048 píxeles, ya que el hardware no podía realizar el procesamiento, inclusive cuando modificábamos el hardware hasta 8 vCPUs, 30 GB de RAM y dos tarjetas gráficas. De igual modo, en el modelo GAN no se puedo realizar el paso de 512x512 hasta 1024x1024 debido a que la memoria de la GPU no era suficiente.

Para la elección de un buen *learning rate*, nos basamos en las recomendaciones de uno de los colaboradores de FastAI [29].

La librería fastai nos permite graficar los valores de pérdida frente al *learning rate*. En la imagen 28 podemos observar este gráfico.

En un principio podemos afirmar que un buen *learning rate* es un valor cercano a 1e-2, ya que es donde se encuentra el valor mínimo del valor de pérdida. Sin embargo, este *learning rate* es un valor alto, ya que se encuentra en el límite entre mejorar y empeorar. Un buen valor sería aquel que tiene un orden de magnitud menor. Por ejemplo, en la imagen posterior, un buen *learning rate* sería 1e-3 en lugar de 1e-2.

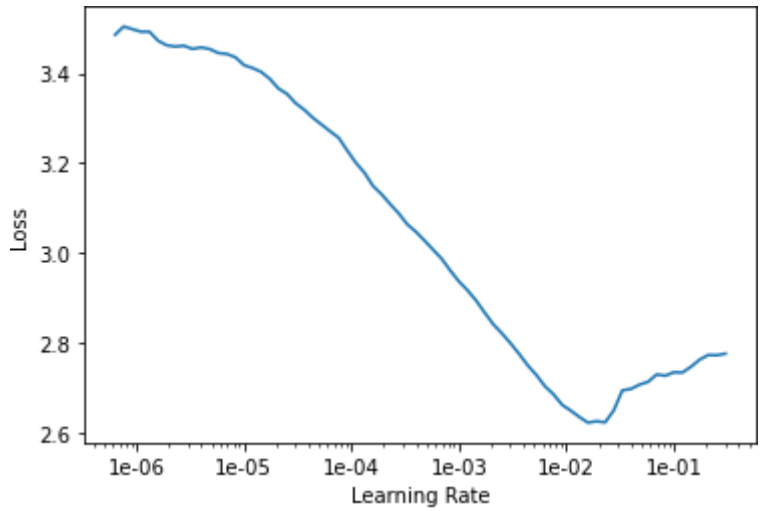

Imagen 28: *Learning rate* de un modelo

Sin embargo, es necesario tener en cuenta la elección de un buen *learning rate* puede complicarse según el gráfico que se nos presenta. Muchas veces, la elección no puede ser tan clara como en la imagen anterior. La única solución es simplemente probar diferentes valores hasta que encontremos uno que nos satisfaga. Incluso algunos valores que a priori pueden ser buenos según la interpretación de la gráfica pueden llevarnos a resultados no tan buenos.

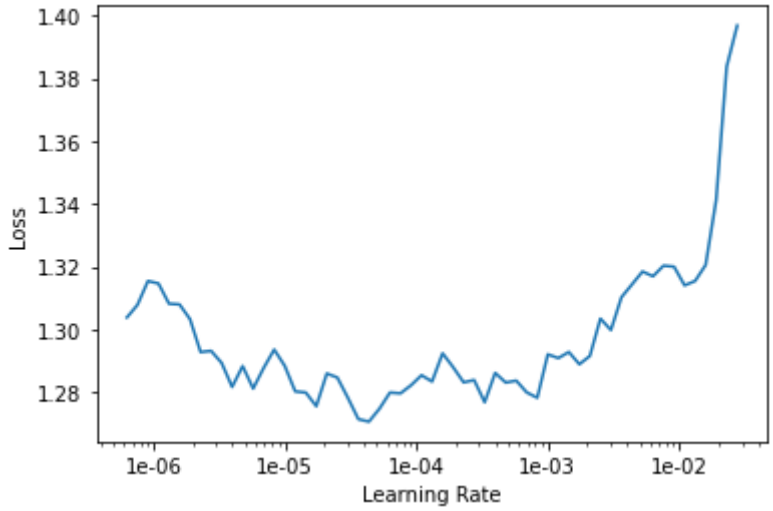

Imagen 29: *Learnig rate* con mayor dificultad de elección

Por otro lado, durante la evaluación de los tres primeros modelos, tanto en el modelo con Píxel MSE como en el modelo GAN, la evaluación producía muchos errores porque se necesitaba más memoria GPU de la disponible. En un primer momento, pensamos que se trataba de un problema ocurrido en el entrenamiento, por lo cual se entrenaron más modelos. Sin embargo, el problema continuaba. Incluso, probamos a entrenar un modelo de prueba con un dataset de 11 imágenes, sin ningún resultado satisfactorio.

A continuación, mostramos el error descrito anteriormente. Este era el problema principal que nos encontrábamos durante todo el entrenamiento y evaluación.

```
RuntimeError: CUDA out of memory. Tried to allocate 71.07 GiB (GPU 0; 14.73 GiB total capacity; 4.26 GiB already allocated;
9.38 GiB free; 4.64 GiB reserved in total by PyTorch)
```

Imagen 30: Error por falta de memoria gráfica

Debido a este problema, la evaluación de los tres modelos lo realizamos prediciendo el dataset de testeo hasta una resolución de 1024x1024 píxeles, para luego reducir las imágenes originales a esta resolución y poder compararlas.

Sin embargo, este problema no ocurría con el modelo con *perceptual loss.* Por lo cual, elegimos a este modelo para realizar el modelo con el segundo dataset de entrenamiento, sin tener mucho en cuenta la evaluación anterior.

Una vez elegido el modelo, en el entrenamiento del modelo con el segundo dataset era necesario añadir otro paso intermedio, ya que si entrenábamos al modelo desde 512x512 hasta 1024x1024 se producía sobreajuste, inclusive con valores bajos de *learning rate*. Por lo cual, añadimos un paso extra de 512x512 hasta 768x768, y luego de 768x768 hasta 1024x1024, lo cual reducía el sobreajuste.

En la imagen posterior podemos ver como en el modelo se producía sobreajuste, la pérdida de entrenamiento empezaba a disminuir frente a la pérdida de validación.

epoch	train_loss	valid_loss	pixel	feat_0	feat_1	feat_2	gram_0	gram_1	gram_2	time
0	1.321402	1.306903	0.277629	0.272382	0.268307	0.085328	0.187496	0.170447	0.045314	51:08
1	1.322075	1.299506	0.280612	0.274271	0.268357	0.086201	0.181993	0.163848	0.044225	51:01
2	1.251800	1.309124	0.277450	0.272253	0.266245	0.083618	0.193083	0.169912	0.046562	51:14
3	1.326261	1.311211	0.277235	0.271154	0.266687	0.087086	0.196345	0.168621	0.044083	51:13
4	1.287645	1.337215	0.275250	0.274977	0.270445	0.086075	0.210567	0.174000	0.045902	51:19
5	1.313588	1.356610	0.279561	0.277066	0.272763	0.085866	0.213908	0.180212	0.047233	51:16
6	1.290824	1.368582	0.286238	0.275672	0.270176	0.083724	0.218543	0.184789	0.049441	51:21
7	1.302307	1.352564	0.274230	0.277127	0.272439	0.089041	0.214506	0.179763	0.045459	51:25
8	1.285531	1.322950	0.278860	0.273260	0.263204	0.086036	0.202071	0.174786	0.044732	51:22

Imagen 31: Sobreajuste del modelo

En cuanto a la evaluación, en estos dos últimos modelos realizamos la evaluación planteada inicialmente en el diseño.

Un aspecto para tener en cuenta es que para cada imagen con la realizábamos inferencia, teníamos que cargar de nuevo el modelo. El motivo, es el parámetro *size* que mostramos en la imagen 32.

Este parámetro modifica la resolución de entrada y salida del modelo. Debido a que cada imagen con la que intentamos aumentar su resolución tiene una resolución diferente, la memoria de la GPU se llena rápidamente, ya que, al modificar este parámetro, no se vacía la memoria automáticamente. Por ello, es necesario vaciar la memoria de la tarjeta gráfica.

La única solución encontrada fue eliminar el modelo y vaciar manualmente la memoria. Este proceso lo podemos ver en la imagen 33.

```
learn = load_learner(path, 'perceptual2.pkl')
data = (src.label_from_func(lambda x: path_hr/x.name)
    .transform(None, size=size, tfm_y=True)
    .databunch(bs=1).normalize(imagenet_stats, do_y=True))
data.c = 3
learn.data = data
```

Imagen 32: Parámetro *size*

```
learn = None
gc.collect()
torch.cuda.empty_cache()
```

Imagen 33: Vaciado de la memoria

5.3. Programa

Por último, en cuanto al programa, lo único que teníamos que hacer era refactorizar el código y pasarlo a un fichero con extensión .py.

El programa admite cualquier modelo ya entrenado y exportado para realizar inferencia (función export de fastai). Para ello, es necesario suministrar al programa con un directorio donde se encuentren las imágenes a aumentar de resolución, un directorio donde se guardarán las imágenes generadas por el modelo y la resolución a la que queremos aumentar las imágenes.

Cabe destacar que, para que el programa funcione, es necesario tener instalado Pytorch y FastAI.

De igual manera, cuanto más grande sea la resolución que deseemos obtener de las imágenes, más cantidad de memoria de la tarjeta gráfica es necesaria.

Asimismo, los modelos con Píxel MSE y modelos GAN presentan inconvenientes que no permiten aumentar las imágenes a una resolución alta (la máxima resolución probada que funcionaba era 1024x1024 píxeles).

En la imagen 34 podemos observar los parámetros del programa al que hemos llamado superresolution.

```
jupyter@pytorch-20200516-200303:~$ ./superresolution.py -h
usage: superresolution.py [-h] -in PATH_LR -out PATH_GEN -l LEARNER WIDTH HEIGHT

Aumenta la resolución de una ortofoto aérea.

positional arguments:
  WIDTH          anchura
  HEIGHT         altura

optional arguments:
  -h, --help     show this help message and exit
  -in PATH_LR    Directorio con las imágenes
  -out PATH_GEN  Directorio de salida.
  -l LEARNER     Modelo (extensión .pkl). Tiene que estar dentro del directorio de entrada.
```

Imagen 34: Programa

6. Evaluación

A continuación, mostramos la evaluación de los diferentes modelos según las métricas PSNR y SSIM, y el tiempo de inferencia. Además, presentamos una sección de experimentación para mostrar los límites de los modelos.

6.1. Resultados

A continuación, comparamos las imágenes predichas por los diferentes modelos con el dataset del testeo frente a sus imágenes originales mediante las métricas PSNR y SSIM, además, del tiempo que tarda cada modelo en predecir todo el dataset.

Cuanto mayor sea el valor de PSNR y más cercano sea el valor de SSIM a uno, mejor habrá sido la reconstrucción de la imagen. Igualmente, cuanto más bajo sea el tiempo, más eficiente habrá sido el modelo al predecir el dataset.

La tabla 10 muestra los resultados de la primera evaluación de los modelos.

Modelo	PSNR	SSIM	Tiempo
PMSE	15.3324	0.2224	20min 6s
GAN	15.7996	0.2080	21min 2s
Perceptual Loss 1	14.8536	0.17326	20min 6s

Tabla 10: Evaluación de los primeros modelos

La tabla 11 muestra los resultados de los modelos con la evaluación inicialmente diseñada.

Modelo	PSNR	SSIM	Tiempo
Perceptual Loss 1	19.8826	0.2761	35min 58s
Perceptual Loss 2	19.8989	0.2808	36min 31s

Tabla 11: Segunda evaluación de los modelos

Podemos observar como en la tabla 10 el modelo con *perceptual loss* produce los peores resultados tanto en PSNR como en SSIM. Sin embargo, debido a las limitaciones de esa evaluación, no podemos asegurar que este modelo sea el peor.

Por otra parte, la tabla 11 nos muestra como el segundo modelo con *perceptual loss* es ligeramente mejor en las métricas PSNR y SSIM. Esto nos puede indicar que este segundo modelo es más robusto y produce mejores resultados.

En cuanto a los tiempos de evaluación, todos los modelos producen tiempos similares, lo cual nos lleva a asegurar que el tiempo de inferencia, en nuestro caso, no es importante.

6.2. Comparación visual

A continuación, comparamos visualmente diferentes secciones de las imágenes predichas por los modelos frente a su imagen original. La imagen predicha tiene la misma resolución que la imagen original.

De izquierda a derecha, la imagen a predecir, la imagen predicha con el modelo *perceptual loss* 1, la imagen predicha con el modelo con *perceptual loss* 2 y la imagen original.

En este primer ejemplo, la imagen a predecir tiene una resolución de 714x494 píxeles y la imagen original tiene una resolución de 3570x2470 píxeles.

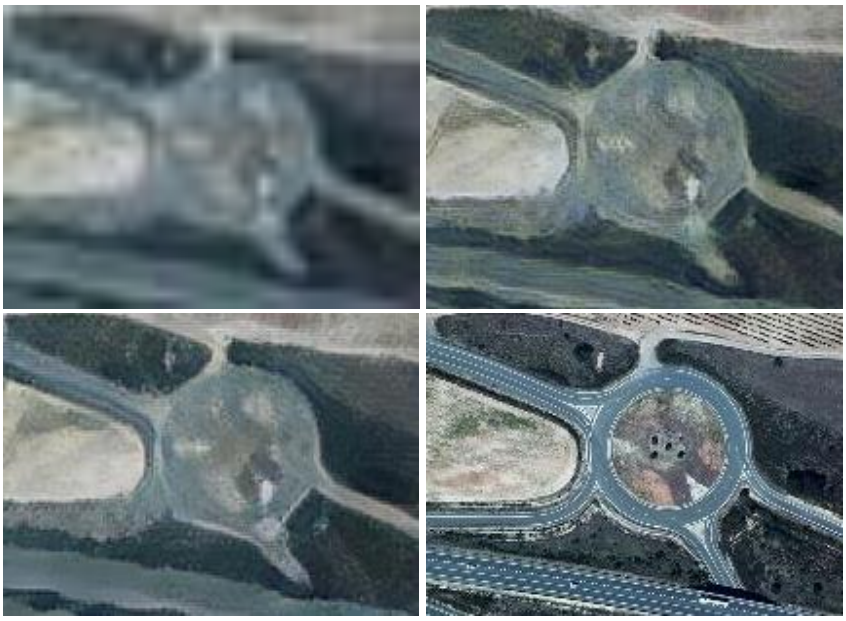

Imagen 35: Comparación visual nº1

En este segundo ejemplo, la imagen a predecir tiene una resolución de 710x490 píxeles y la imagen original tiene una resolución de 3550x2450 píxeles.

Imagen 36: Comparación visual nº2

6.3. Experimentación

A continuación, realizamos inferencia con diversas imágenes recolectadas de Internet.

En este tercer ejemplo, la imagen a predecir tiene una resolución de 710x490 píxeles y la imagen original tiene una resolución de 1702x1168 píxeles. La imagen original fue recolectada de la página web del Ayuntamiento de Ermua [41].

Imagen 37: Comparación visual nº3

En este cuarto ejemplo, la imagen a predecir tiene una resolución de 99x76 píxeles y la imagen original tiene una resolución de 1140x856 píxeles.

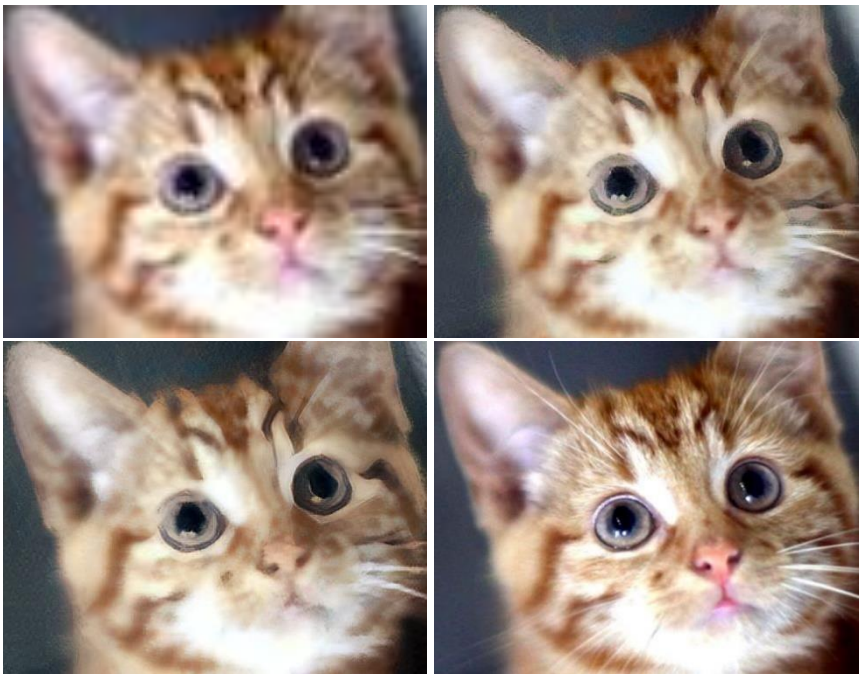

Imagen 36: Comparación visual nº4 [42]

En este quinto ejemplo, la imagen a predecir tiene una resolución de 121x98 píxeles y la imagen original tiene una resolución de 1280x720 píxeles.

Imagen 37: Comparación visual de un fotograma de Apocalypse Now

Con estos últimos dos ejemplos podemos observar que ambos modelos no pueden reconstruir de manera adecuada cada imagen, lo cual es de esperar, pues los modelos no han sido entrenados para este tipo de imágenes.

7. Seguimiento y control

Esta sección está dedicada al seguimiento y control realizado a lo largo del proyecto.

7.1. Riesgos

A pesar de la planificación inicial, durante el desarrollo del proyecto ocurrieron algunos de los riegos planteados en ella.

Uno de estos riesgos, pérdida de la memoria, ocurrió por guardar la memoria en el mismo directorio donde trabajamos con las imágenes durante la etapa del desarrollo con Google Colab y Google Drive. Durante esta etapa, llenamos Google Drive de muchas imágenes. Por lo cual, era necesario vaciarlo para dar espacio a nuevas imágenes. Para dejar ese espacio libre teníamos que eliminar permanentemente los archivos. De este modo, eliminamos por accidente la memoria de forma permanente. Este accidente nos hizo desviar de la planificación inicial y no lograr conseguir los hitos de las etapas posteriores.

Asimismo, durante el proyecto surgió la necesidad de cambiar de plataforma. Aunque teníamos conocimiento de Google Cloud, no teníamos conocimiento de los servicios necesarios para trabajar en nuestro proyecto. De igual manera, este inconveniente solo nos produjo un pequeño retraso que no llevó a ningún problema con el cumplimiento de la planificación.

7.2. Estimaciones de dedicación

La tabla 12 incluye la dedicación planificada y la real para cada paquete de trabajo y para el proyecto completo. Se incluyen los porcentajes de desviación al final del proyecto.

Paquete de trabajo		Plan	Real	Desviación
TFG-1.1	Planificación	30h	20h	-33.3%
TFG-1.2	Seguimiento y control	20h	10h	-50%
TGF-2.1	Análisis	30h	30h	0%
TFG-2.2	Diseño	30h	30h	0%
TFG-2.3	Entrenamiento	120h	125h	+4.17%
TFG-2.4	Evaluación	10h	15h	+50%
TFG-3	Memoria	60h	80h	+25%
TFG		300	310	+3.33%

Tabla 12: Dedicación planificada y real

7.3. Cronograma real

La tabla 13 contiene el Gantt del proyecto con los periodos de realización planificados (barras amarrillas) y reales (barras grises) de cada paquete de trabajo.

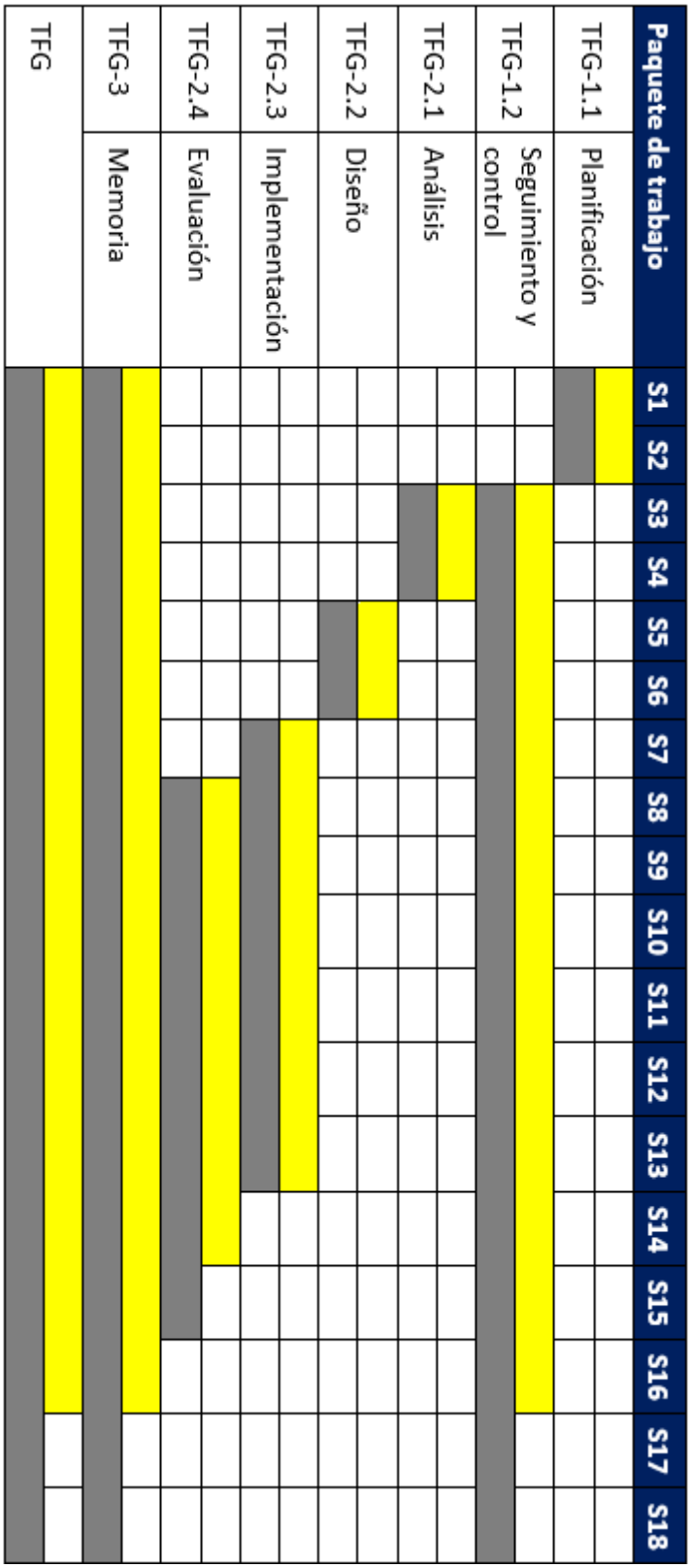

Tabla 13: Diagrama de Gantt

7.4. Entregables

La tabla 14 muestra los entregables generados en el proyecto, los cuales están disponibles en el siguiente repositorio de GitHub: https://github.com/chbenalc/TFG

Producto	Subproducto	Nombre
Jupyter Notebooks	GAN	GAN.ipynb
	PMSE	PMSE.ipynb
	Perceptual Loss 1	Perceptual_loss.ipynb
	Perceptual Loss 2	Perceptual_loss2.ipynb
Modelos	GAN	GAN.pkl
	PMSE	PMSE.pkl
	Perceptual Loss 1	Perceptual1.pkl
	Perceptual Loss 2	Perceptual2.pkl
Programa		Superresolution.py

Tabla 14: Entregables del proyecto

8. Conclusiones

Durante el desarrollo de este proyecto nos hemos encontrado con diferentes problemas tecnológicos que han impedido la mejora de los diferentes modelos de superresolución.

Algunos de estos problemas resultaban difíciles de solucionar, porque mucha de la información no estaba disponible en un solo lugar, sino que estaba dispersada por todo Internet, especialmente en foros de discusión.

Además, uno de los requisitos, que todos los modelos sean capaces de aumentar en cinco veces la resolución de una imagen, solo ha sido completado parcialmente, ya que solo dos de los cuatros modelos creados son capaces de hacerlo. Todos los modelos presentan capacidad de mejora.

Sin embargo, este proyecto ha significado en una experiencia enriquecedora tanto a nivel personal como a nivel profesional. A nivel personal, ha significado tener que enfrentarme a las consecuencias de mis decisiones, por ejemplo, al tener una confianza excesiva durante la realización del proyecto, nos ha llevado a encontrarnos con problemas que podrían haber evitado con facilidad.

Asimismo, a nivel profesional, este trabajo ha supuesto en una gran fuente de aprendizaje, no solo en el ámbito de la visión por ordenador, sino también en una mejora en la capacidad de resolución de problemas. El conocimiento adquirido con este proyecto me ha servido para conocer mejor la dirección que deseo seguir con mi educación y las áreas en las que me gustaría desarrollar mi carrera profesional, la cual estaría relacionada con la inteligencia artificial.

La superresolución es un campo muy interesante debido a los espectaculares resultados obtenidos en otros proyectos, como la restauración de películas antiguas [43] [44] y por su amplia gama de aplicación a distintos proyectos.

Bibliografía

[1] Instituto Geográfico Nacional, «Geoportal SIOSE,» [En línea]. Available: https://www.siose.es/.

[2] Instituto Geográfico Nacional, «SIOSE Alta Resolución,» [En línea]. Available: https://www.siose.es/web/guest/siose-alta-resolucion.

[3] Gobierno de La Rioja, «IDERioja,» [En línea]. Available: https://www.iderioja.larioja.org/index.php?id=1&lang=es.

[4] Gobierno de La Rioja, «Folleto IDERioja,» [En línea]. Available: https://www.iderioja.larioja.org/archivos/pdf/iderioja_es.pdf.

[5] R. A. J. PINOCHET, «IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE SUPER-RESOLUCIÓN PARA IMÁGENES TOMADAS CON NANO SATÉLITES,» [En línea]. Available: http://repositorio.uchile.cl/bitstream/handle/2250/164017/Implementaci%C3%B3n-y-evaluaci%C3%B3n-de-algoritmos-de-super-resoluci%C3%B3n-para-im%C3%A1genes-tomadas-con-nano-sat%C3%A9lites.pdf?sequence=1&isAllowed=y.

[6] B. Hamm, «All About Images,» [En línea]. Available: https://guides.lib.umich.edu/allaboutimages.

[7] S. Patterson, «Image Resolution, Pixel Dimensions and Document Size in Photoshop,» [En línea]. Available: https://www.photoshopessentials.com/essentials/image-resolution/.

[8] T. Thomas, «How To Prepare for The High-Resolution Web,» [En línea]. Available: https://medialoot.com/blog/high-resolution-web/.

[9] B. Sahu, «An Evolution in Single Image Super Resolution using Deep Learning,» [En línea]. Available: https://towardsdatascience.com/an-evolution-in-single-image-super-resolution-using-deep-learning-66f0adfb2d6b.

[10] MathWorks, «Nearest Neighbor, Bilinear, and Bicubic Interpolation Methods,» [En línea]. Available: https://es.mathworks.com/help/vision/ug/interpolation-methods.html.

[11] BBVA, «'Machine learning': ¿qué es y cómo funciona?,» [En línea]. Available: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/.

[12] L. D. a. D. Yu, «Deep Learning. Methods and Applications,» [En línea]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/DeepLearning-NowPublishing-Vol7-SIG-039.pdf.

[13] J. Cowley, «Redes neuronales convolucionales,» [En línea]. Available: https://www.ibm.com/developerworks/ssa/library/cc-convolutional-neural-network-vision-recognition/index.html.

[14] F. Ramírez, «Las matemáticas del Machine Learning: Redes Neuronales (Parte I),» [En línea]. Available: https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-redes-neuronales-parte-i/.

[15] G. A. G. e. al., «ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL USANDO EL ALGORITMO,» [En línea]. Available: https://dialnet.unirioja.es/descarga/articulo/4844874.pdf.

[16] J. C. S. C. H. F. I. Zhihao Wang, «Deep Learning for Image Super-resolution: A Survey,» [En línea]. Available: https://arxiv.org/pdf/1902.06068.pdf.

[17] ImageAnnotation.AI, «Single Image Super-Resolution Through Deep Learning: The Power of GANs,» [En línea]. Available: https://www.imageannotation.ai/blog/deep-learning-for-single-image-super-resolution.

[18] X. Z. Y. T. W. W. J.-H. X. Q. L. Wenming Yang, «Deep Learning for Single Image Super-Resolution: A Brief Review,» [En línea]. Available: https://arxiv.org/pdf/1808.03344.pdf.

[19] P. F. a. T. B. Olaf Ronneberger, «U-Net: Convolutional Networks for Biomedical Image Segmentation,» [En línea]. Available: https://arxiv.org/pdf/1505.04597.pdf.

[20] K. Ahirwar, «Generative Adversarial Networks Projects,» [En línea]. Available: https://www.oreilly.com/library/view/generative-adversarial-networks/9781789136678/de6e3e20-2a2f-4bf5-b304-19d719f540a3.xhtml.

[21] A. A. L. F.-F. Justin Johnson, «Perceptual Losses for Real-Time Style Transfer and Super-Resolution,» [En línea]. Available: https://arxiv.org/abs/1603.08155.

[22] K. S. a. A. Zisserman, «VERY DEEP CONVOLUTIONAL FOR LARGE-SCALE IMAGE RECOGNITION,» [En línea]. Available: https://arxiv.org/pdf/1409.1556.pdf.

[23] Stanford Vision Lab, Stanford University, Princeton University, «ImageNet,» [En línea]. Available: http://www.image-net.org/.

[24] J. Howard, «Lesson 7: Resnets from scratch; U-net; Generative (adversarial) networks,» [En línea]. Available: https://github.com/hiromis/notes/blob/master/Lesson7.md.

[25] W. Fulton, «What does JPG Quality Losses mean? What are JPG artifacts?,» [En línea]. Available: https://www.scantips.com/basics09b.html.

[26] fast.ai, «vision.transform,» [En línea]. Available: https://docs.fast.ai/vision.transform.html.

[27] C. Thomas, «Deep learning based super resolution, without using a GAN,» [En línea]. Available: https://towardsdatascience.com/deep-learning-based-super-resolution-without-using-a-gan-11c9bb5b6cd5.

[28] H. Zulkifli, «Understanding Learning Rates and How It Improves Performance in Deep Learning,» [En línea]. Available: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

[29] S. Gugger, «How Do You Find A Good Learning Rate,» [En línea]. Available: https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html.

[30] fast.ai, «vision.learner,» [En línea]. Available: https://docs.fast.ai/vision.learner.html.

[31] J. S. a. Q. V. L. Simon Kornblith, «Do Better ImageNet Models Transfer Better?,» [En línea]. Available: https://arxiv.org/pdf/1805.08974.pdf.

[32] R. G. P. D. Z. T. K. H. Saining Xie, «Aggregated Residual Transformations for Deep Neural Networks,» [En línea]. Available: https://arxiv.org/pdf/1611.05431.pdf.

[33] B. Raj, «An Introduction to Super Resolution using Deep Learning,» [En línea]. Available: https://medium.com/beyondminds/an-introduction-to-super-resolution-using-deep-learning-f60aff9a499d.

[34] L. L. a. A. C. B. Zhou Wang, «Video Quality Assessment Based on Structural Distortion Measurement,» [En línea]. Available: https://ece.uwaterloo.ca/~z70wang/publications/vssim.pdf.

[35] fast.ai, «fastai,» [En línea]. Available: https://github.com/fastai/fastai.

[36] Project Jupyter, «Jupyter,» [En línea]. Available: https://jupyter.org.

[37] Google, «Colaboratory,» [En línea]. Available: https://research.google.com/colaboratory/faq.html.

[38] Google, «Google Cloud Plataform,» [En línea]. Available: https://cloud.google.com/free/docs/gcp-free-tier?hl=es.

[39] Amazon Web Services, «Capa gratuita de AWS,» [En línea]. Available: https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc.

[40] Amazon Web Services, «AWS Educate,» [En línea]. Available: https://aws.amazon.com/es/education/awseducate/students/.

[41] A. d. Ermua, «Ortofoto 63-57 E 1:5000,» [En línea]. Available: https://www.ermua.es/pags/urbanismo/ca_ortofotos_6357.asp.

[42] K. Brulliard, «Cat experts reveal the meaning behind different meows,» [En línea]. Available: https://www.independent.co.uk/news/science/cat-experts-reveal-the-meaning-behind-different-meows-a6895251.html.

[43] D. Shiryaev, «[4k, 60 fps] Arrival of a Train at La Ciotat (The Lumière Brothers, 1896),» [En línea]. Available: https://www.youtube.com/watch?v=3RYNThid23g.

[44] D. Shiryaev, «[4k, 60 fps] A Trip Through New York City in 1911,» [En línea]. Available: https://www.youtube.com/watch?v=hZ1OgQL9_Cw.